TOP TIPS FOR WEB-DEPLOYED FORMS

Peter Koletzke, Quovera

Since early versions of Forms 4.5, Oracle has offered the option to deploy applications on the Web. Companies find the benefits of this technology are compelling. This strategy offers the centralized location of application code that provides ease of maintenance, ease of deployment, and ease of administration. It also avoids the desktop installation problems that long plagued client/server environments. Add to this the maturity of the Oracle solution and the growing web awareness of user and business communities, and the choice of web-deployed forms is clear for many Oracle customers.

When you deploy Oracle Forms Developer applications on the Web you will find that there are some things that work and some things that do not work the same as on client/server. Not only is the architecture different, but you will find that your design and coding require modifications if you are accustomed to deploying to client/server. It is useful, therefore to have at hand some tips for working in this environment.

This paper discusses a number of categories of these tips. Some are key pieces of information. Others are lower-level technical details. The following are the baker's dozen categories of tips in this paper:

1. **Understand the architecture**
2. **Read the documentation**
3. **Design for the Web**
4. **Tune for the Web**
5. **Test on the Web**
6. **Move code to the database**
7. **Set user expectations**
8. **Use the "Oracle look and feel"**
9. **Use a forms shell**
10. **Understand operating system access**
11. **Know the limitations and workarounds**
12. **Troubleshoot problems efficiently**
13. **Don't go it alone**

The paper addresses work within a Forms Developer 6i environment. As of this writing, Oracle9i Developer Suite (that includes Forms 9i) has not been released into production, but based upon prerelease information, the tips discussed in this paper should apply to that environment as well.

Note

Forms 9i will not support deployment to client/server environments. Therefore, the discussions in this paper about client/server apply mainly to those who are migrating forms to the Web or are working in a Forms 6i environment.

1. UNDERSTAND THE ARCHITECTURE

Although this may not fit into your standard expectation for a tip, it is the most important of all the tips. Understanding the architecture thoroughly is the basis for effective work in this environment. A brief review of this architecture appears in Appendix A of this paper. Therefore, the tip consists of reading that discussion to ensure that you understand the environment in which you are working.

2. READ THE DOCUMENTATION

There is documentation that can assist in your search for more information on this new environment. Some of the best sources of information are documents and technical white papers from Oracle. User and vendor conferences have produced a number of white papers containing user experiences and advice with the product. Previous IOUG conferences as well as the ODTUG and Oracle OpenWorld conferences have presented more white papers on web-deployed forms. Search the user group web sites mentioned in category 13 below for these papers. Oracle's Technology Network (OTN) web site (otn.oracle.com) also contains many essential white papers on the subject. Look in the product area and browse the Forms and Forms Server categories.

The Oracle Information Navigator (OIN) is available as "Oracle Forms Developer Online Manuals" in the Windows Start menu group Oracle Forms 6i and also from the Form Builder **Help→Manuals** option. OIN contains the following manuals:

- ***Deploying Forms Applications to the Web with Forms Server*** This HTML manual contains the essential Chapter 2, *Overview of Forms Services*. This contains an excellent description of the architecture, details on how to set up and troubleshoot the server. The other chapters in this manual are equally helpful and include subjects such as configuration and troubleshooting. It is worth the time to read or skim this entire manual.
- ***Guidelines for Building Applications*** Chapter 3, *Performance Suggestions*, explains general tuning techniques for Forms applications and provides a specific section for special performance enhancements in a three-tier architecture.
- ***Release Notes*** This file contains information that is not included in other sources. It is useful to check here for known limitations or workarounds to Forms features.
- ***Form Builder Reference*** This manual is available in the Reference Manuals node. It contains all reference information about the Form Builder objects, properties, built-ins, variables, and extensions. When creating a startup HTML file, you need to know the Forms command-line parameters (found in the topic "Forms Runtime Options").
- **New Features** This document is worth reading to understand what has been added to the product recently.

Most manuals available in the OIN are installed into the ORACLE_TOOLS_HOME/tools/doc60/guide60/us (although other languages may install into a different subdirectory of GUIDE60). Some have Adobe acrobat (.PDF) versions in addition to the HTML versions that OIN manages.

WHITE PAPERS ON THE FORMS SERVER

There is a paper that this author copresented at Oracle OpenWorld 2000 about the Forms Server setup and troubleshooting in an OAS 4 environment; some of the same considerations and techniques apply to Oracle9iAS. The company website mentioned in the author biography at the end of the paper contains a copy of this paper.

3. DESIGN FOR THE WEB

You cannot always deploy on the Web the same forms that you develop for client/server. Since the runtime environments are different, it is not practical to design a form for client/server and expect to run it the same way on the Web. Therefore, if your deployment environment will be the Web, you have to design for it. The main design concerns are in how the Forms runtime system handles the visual elements and mouse actions. There are additional concerns in optimizing the forms to minimize network traffic from the client to the application server (mentioned later in this paper). If you are migrating forms from client/server to the Web, some of the design considerations will help you determine what you may need to change.

Developers who are accustomed to client/server development need to remember that the application server runs the forms and presents only the interface on the client. Application code executes on the application server, but the user interface objects such as text items and poplists will generate network traffic anytime the user acts on the object. There is no network traffic created if the user is just typing into a text item, but any navigation out of the item will create network traffic as a set of triggers fire.

One key success factor to a project that uses web-deployed forms is being sure that you can implement the design using this technology. Therefore, in client/server applications or even in web applications with other products, you may want to hire a GUI designer to analyze and create an interface design. With web-deployed forms, this can be a key failure factor, as the designer can easily ignore the many limitations of the technology in favor of a friendly and pretty user interface.

You may need to make some compromises in your design standards, if you have them, for client/server forms deployed on the Web. For example, while many client/server Forms applications are built to exercise the Windows operating system and provide feature-rich user interfaces, you may need to consider simplifying the interface for the Web.

WEB DESIGN ELEMENTS

In addition, some design elements may need to be adjusted if your intended audience is accustomed to a certain style of web application. For example, if your users are accustomed to a standard HTML interface, the standard client/server menus and toolbars may be foreign to them. Your interface may be more intuitive to that audience if it uses canvas buttons to accomplish the required functions.

Another element that you need to decide upon is how the user will query records. The native Forms' Enter Query mode may not suffice for users accustomed to web applications. They may be more comfortable with a search box and results page (as they are used in search engine web sites) and you can adjust your form design to accommodate this technique.

DIFFERENT KEY MAPPINGS

The default key mappings are different between client/server and web forms. The standard file used in web-deployed forms to map keypresses to Forms functions is FMRWEB.RES (located in the FORMS60 directory). This contains key mappings for the Solaris environment. There is another file, FMRPCWEB.RES (in the same directory), that maps keypresses to Forms functions using the same keys as in client/server. Using that file (by renaming FMRPCWEB.RES to FMRWEB.RES) instead of FMRWEB.RES will provide you with the same key mappings on the Web and client/server. Both are plain ASCII text files that you can edit.

Note

Read Chapter 5—Designing Portable Applications—of the *Guidelines for Building Applications* online manual (**Help→Manuals** in Form Builder). This chapter contains hints on how to consider platform-specific elements when designing forms. It lists platform-specific restrictions for Windows and Motif. If you need to support web-deployed forms and client/server forms, this kind of advice is invaluable.

SIMPLIFY YOUR FORMS

Forms that have a small number of GUI items work very well. In contrast, forms with many items on one canvas can be problematic. Performance is reasonable with a limited number of users (based on the power of the application server) and a dedicated server with a basic level of memory (3–10 megabytes (MB) per concurrent user in R.2 and 1–6MB in R.6). Oracle has tested memory usage on the application server of a small form to be 1–2MB per user. They can support between 100 and 500 users per CPU and have proven the viability of web-deployed forms with up to 5,900 concurrent users.

SIMPLE SAMPLE

The author worked on a project where the startup form appeared 5 seconds after the user clicked on the URL. This form was a simple startup form such as that shown in Figure 1. This form was used only to accept the user input of user ID and password (and to change the password). After it successfully logged into the database, it called the first menu form for the application. Since this form had very few graphical elements and Forms' objects, the startup time was minimized. Consider making your starting form as lightweight as this type of form.

Caution

If you use the strategy of creating many forms instead of one large form, the initial startup of each form will be faster, but moving from one form to another may be slower because a new form needs to start up. If the functions on the other form are rarely used, dividing the form may be better. Most of the time, the form would load quickly. But when you call another form, there would be a longer delay than usual, because a new form is starting. This wait may be less objectionable to users than the initial startup wait, so this strategy may prove useful.

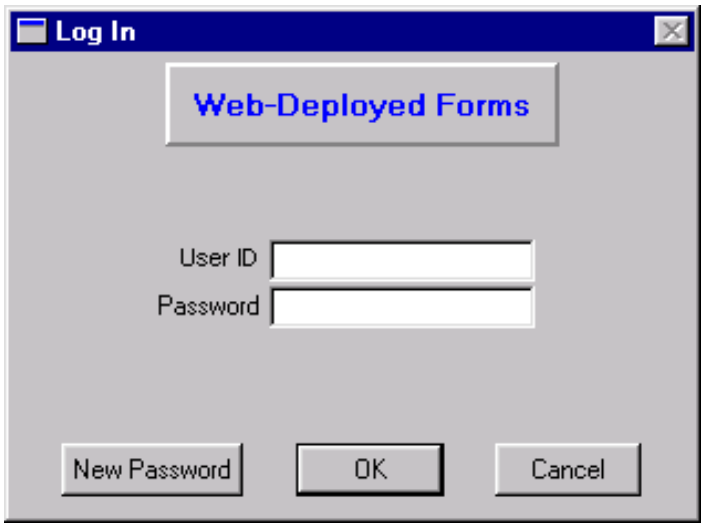

Figure 1. Simple startup form

FONTS

It is not reasonable to expect Forms to use the same fonts, or in some cases, colors, in client/server and the Web—these are different environments. The differences in fonts could cause sizing issues; the difference in colors could cause unexpected displays. Both problems are solvable by reworking the form in Form Builder. Java only offers five fonts: Dialog, Serif, SansSerif, Monospace, DialogInput. None of these are available in Windows so that the Forms runtime must map any font you have used to its Java equivalent. You can also change the registry.dat file, which contains font aliases from Java fonts. In R.6, this file is located in the ORACLE_HOME/forms60/java/oracle/forms/registry directory. Font sizing and aliasing is documented in the *Deploying Forms Applications to the Web with Forms Server* chapter on "Application Design Considerations."

> **Note**
>
> The standard MS Sans Serif font used in many Windows applications becomes the Dialog font in web-deployed forms. Arial becomes Helvetica. Both web fonts are different from the Windows fonts in size and shape, but are close approximations. Courier New and Times New Roman fonts in Windows translate to similarly named fonts on the Web. Oracle recommends that you use 9 point MS Sans Serif for building normal Forms items and prompts. This font maps best to the standard Java font "Dialog."

If a font size that you assign to boilerplate text does not render properly, use a prompt property of an item or use a display item. These behave better in earlier versions of Forms 6i.

BUTTON COLORS

On client/server, buttons are usually gray by default because the Windows Control Panel Colors applet manages the color of all buttons on that machine. In web-deployed forms, the Control Panel Colors applet is not in control, so buttons may be a different color (usually the color of the canvas). However, you do have control over the button colors. Create a visual attribute and attach it to the buttons. Better still, create a button SmartClass with the visual attribute attached to it in the object library, and subclass the button using that SmartClass. This technique must work into your design.

STARTUP PARAMETERS

You can add parameters to the startup HTML file to take advantage of some web-only features. The following is list of some of these parameters. For examples and details on the startup file, consult the online documentation (**Help→Manuals**). These examples use the Java lowercase naming convention, but Forms is not case sensitive.

- **useSDI=YES** The default for this parameter is "NO" which means that there will be a multiple document interface (MDI) where all windows appear in a single outer window. If you do not want the MDI window, use the YES value for

this parameter. The MDI window is probably more familiar to users so you would normally not need to use this parameter.

- **separateFrame=TRUE** This value will open a separate window for the form that is outside of the browser frame. The default of FALSE is probably more standard to users so you would not normally need to specify this.
- **splashScreen=co_logo.gif** This defines a splash page graphic that displays when the forms runtime is starting. This gives the user something to look at while the applet is loading and can help the user's favorable perception of the load time.
- **lookAndFeel=oracle** This specifies the "Oracle look and feel" which is an alternative appearance for the Forms applet. This is discussed further in category 8 below. The parameter, ColorScheme, specifies the colors that will be used for this applet. An additional parameter, DarkLook (for example, DarkLook=TRUE) will use darker colors for canvases. This makes the text items and input areas stand out more from the background.
- **readOnlyBackground=true** This causes read-only items to be rendered with a light gray background. This signals to the user that they cannot enter or interact with data in the items and is a common interface technique. You do not need to create and attach visual attributes to each item for this to work. The parameter setting is all that is required.

REGISTRY.DAT PARAMETERS

The REGISTRY.DAT file, mentioned above, is a configuration file located on the application server. It contains a number of parameters that affect all forms that it runs. There are a few that allow you to embed web-only functionality into your form:

- **app.ui.lovButtons=true** This causes a button to appear next to each item that has an LOV attached to it. You do not need to write any code or define any button items for this to work.
- **app.ui.requiredFieldVA=true** This parameter sets all required items to a special background color. You do not need to define visual attributes or write code for this to work. You do have to set up the background color that will be used by setting the red-green-blue color in the app.ui.requiredFieldVABGColor parameter, for example, to use a yellow background for required items, the setting would be: **app.ui.requiredFieldVABGColor=255,242,203**.

4. TUNE FOR THE WEB

There are a number of tips that affect the efficiency of your forms and that you should work into your initial design. Reducing network traffic is the name of the game for tuning web-deployed forms. Most of the tuning effort required is between the application server and client, as this is the most complex connection and the one that is most unfamiliar to Forms developers. It is useful to review some guidelines for reducing database server network traffic.

> **Note**
>
> The Oracle white paper (available on OTN) "How To Tune Your Oracle9iAS Forms Services Applications" contains detailed information about how to take advantage of the new application tuning features of Developer R.6.

OPTIMIZE THE CLASS FILE LOADING

The initial load time of the Java applet can be a sore point with users because it seems longer than the form startup in client/server environments. Part of this time is spent in loading the Java class files on the client side. You can optimize this task using the following tips. The following list refers to Java Archive (JAR) files that are collections of the Java classes used to render objects. You can create and open and manage these files using an archive utility such as WinZip. Most classes are named descriptively, so you can check whether a particular function is in a particular JAR file.

- **Load a smaller JAR file** JAR files are cached if you use JInitiator. This helps reduce the initial download time to a minimum. Use the ARCHIVE parameter in your starting HTML file to load one or more JAR files. For example: PARAM NAME="ARCHIVE" VALUE="f60web.jar,icons.jar". The F60WEB.JAR file contains all but the LOV classes and is the one to use if you want to load all common classes when starting up.
- **Tune the cache** The default cache on the client is 20MB. Compare that number with memory as the application is running (using a system monitor such as the Windows NT Task Manager), and ensure that you have enough cache space

available in memory. Increasing physical memory will help if you are running out of memory and swapping to disk. Freeing memory by closing other applications will also help.

- **Locate the JAR file centrally** If JAR files are housed on different application servers in a load-balancing arrangement, the same JAR files could be downloaded from different servers because the classes are cached relative to the server. Therefore, locate the JAR files centrally if you have a load-balancing configuration.
- **Use the deferred-Load feature** A deferred-load feature allows you to embed references to other JAR files within a JAR file. The referenced files will not load immediately, but will wait until the class in that file is required. This deferred loading means that the form can display faster initially, although other classes that are required will be loaded as needed. Oracle supplies a number of JAR files with different contents, but you can create your own JAR files (using an archive utility such as WinZip) that contain references to other files.
- **Store the .GIF icon files in the JAR file**. This saves download time. .GIF files are used for icons on buttons in web-deployed forms instead of .ICO files. Some icons are actually supplied by the Runtime engine and do not require .GIF files at all. There is no known list of these icons, but you can experiment with the icons that are used by the default Forms toolbar (exit, save, etc.). The images will display even if you delete those icons from the icon directory.
- **Store beans in a JAR** If you are using JavaBeans (Pluggable Java Components), store them in the JAR file as well to speed up the download.

Caution

Regardless of the tuning benefits mentioned, creating your own JAR files is a bit difficult and is normally unnecessary. The JAR files supplied with Oracle Forms Developer will serve you well for most situations. In addition, Oracle discourages modification of the supplied JAR files. If you find that you need to modify the contents of a JAR file, it is best to create your own JAR file. That way, if Oracle upgrades a supplied JAR file, you will not have to redo your JAR file work.

REDUCE BANDWIDTH USE

The key factor that differentiates web-deployed forms from client/server forms is their use of the network. Anything you can do to reduce network traffic will speed up the way the forms load or run. The following points address this principle.

- **Use the prompt property** Boilerplate text objects require time to be rendered. The prompt property of an item is faster because it is just a property instead of a separate object. For forms that do not use the prompt property, you can use the Associate prompt button (after group selecting the boilerplate prompt and the item) to move the text to the item property.
- **Reduce the number of boilerplate graphics**. Graphics require extra processing, so the fewer graphics that you use, the better. Also, lines and rectangles are optimized, but other types are not. In addition, Oracle recommends limiting the use of multimedia. Instead of embedding the multimedia file in the form, call a URL that uses a plug-in to display the file.
- **Change navigation** If your form contains many windows, allow the user to navigate to those other windows by clicking an OK or Done button instead of by pressing TAB to navigate through items. If the user does not need to change anything in those items, it is a waste of time to have to navigate to them. In addition, more triggers will fire as you navigate through items (such as POST-TEXT-ITEM and WHEN-NEW-ITEM-INSTANCE). If the user can skip to the next window with a button, the network traffic required by the item triggers will be eliminated.
- **Hide objects not initially required**. Set the canvas property *Raise on Entry* to "Yes." Set other objects' *Visible* property to "No." When the cursor navigates to the item, Forms will automatically display the canvas. You can also issue a SHOW_VIEW call to set the *Visible* property to "Yes" programmatically. Tab canvases load all objects for the entire set of tabs at the same time. Set the *Visible* properties of all items to "No" to counteract this. Set them back to "Yes" when you navigate to a particular tab.
- **Write triggers at the lowest level** You may use form- or block-level item triggers such as a trigger that checks which button was clicked and executes code specific to that button. This method results in code that is easier to maintain

because the developer does not need to drill down many levels to find a particular trigger. However, on the Web, this method will cause extraneous network messages. Each time a trigger fires, a network message is generated. If you have a form-level item trigger (such as WHEN-NEW-ITEM-INSTANCE), that trigger will fire for each item on the form regardless of whether that item requires the logic in the trigger. If the trigger is placed at the lowest level (in this case, an item trigger), the trigger will fire only when needed.

- **Disable menu buffering** Chapter 11 of the *Deploying Forms Applications to the Web Using Forms Server* manual in the Oracle Information Manager of Form Builder (**Help→Manuals**) mentions that menu buffering can help improve performance. Normally, the default for menu buffering is "TRUE." Double check this with your installation by temporarily coding the following in the WHEN-NEW-FORM-INSTANCE trigger

```
MESSAGE(GET_APPLICATION_PROPERTY(MENU_BUFFERING));
```

The message line will show the default value of menu buffering in your application. If it is "FALSE," there is no need to make changes.

When menu buffering is set to TRUE, any change in menu properties (for example, changing a menu item label) requires the entire menu to be resent from the application server to the client when the next "synchronize" event occurs (this may mean when the menu is accessed or "synchronize" is called). Sending the entire menu requires more network access than sending an individual modification. Therefore, if you make few or no modifications to the properties LABEL, ICON, VISIBLE, and CHECKED, turn menu buffering off so that the changes will be immediately made and extra menu downloads will be avoided. Use the following code to disable menu buffering:

```
SET_APPLICATION_PROPERTY(MENU_BUFFERING, 'FALSE');
```

Note

Oracle has implemented "message bundling" that combines similar triggers into one message packet. For example, when the cursor moves from one item to another, a number of triggers fire (such as WHEN-VALIDATE-ITEM and WHEN-NEW-ITEM-INSTANCE). The bundle is unpackaged on the Forms runtime server and the triggers are processed in the normal order. This saves the network overhead that would be generated when multiple messages that would normally be sent when a cursor moves between items.

USE PLUGGABLE JAVA COMPONENTS

Pluggable Java Components (PJCs) are JavaBeans that are either embedded in the form as a separate object (Bean Area on the Layout Editor toolbar) or that act as a substitute for a Forms object such as an item or button. Incorporating PJCs will allow you to extend the capabilities of Forms and solve problems such as how to read and write to the client's file system, how to provide a Java spell checker for a web form, how to implement timers that fire only on the client.

Incorporating Java into your form gives you experience with the Java language and its environment. It also allows you to maintain and leverage your core competency in Forms technology and continue to use Forms for most requirements. As you use Java more and more in your forms, you may find that Java starts to provide a large part of the functionality and you may think about transitioning to a solely Java application. Your experience with PJCs will help your transition into the Java world.

Note

Oracle Technology Network contains examples of PJCs that you can use for file access and a Java spell checker (otn.oracle.com). The author's website mentioned at the end of this paper contains an example of a timer that fires on the client side of a web form and a file dialog that accesses client-side files.

OTHER TUNING TIPS

Here are some other techniques you can use to tune the runtime environment:

- **Avoid repeating timers** Timers generate extra network traffic, as a packet is sent each time that the timer expires. If you destroy the timer and do not allow it to repeat, there is no adverse effect. Instead of using repeating timers, use JavaBeans with the same functionality. JavaBeans will fire on the client side and no extra network traffic will be generated. As mentioned in the previous note, the author's website contains a solution for repeating timers in web forms.
- **Avoid MOUSE-UP and MOUSE-DOWN triggers**, as these require extra processing on the server. Both triggers are handled by the same Java event. Therefore, the forms browser applet sends a message to the Forms Server when a m mouse event occurs and the server determines whether it is a MOUSE-UP or MOUSE-DOWN event. Therefore, if you have a MOUSE-DOWN trigger but no MOUSE-UP trigger, the Java event will occur even in a MOUSE-UP situation. This is extra work for the Forms Server if you are not using that trigger.
- **Optimize validation using PJCs** Item validation generates application server messages and processing. Use PJCs instead of standard Forms objects to do validation. For example, you can replace a text item with a Java item that already contains validation code so that the validation will occur on the client and save network traffic. The extra code may require extra time to load the form, but this disadvantage may be less important than the savings in validation time that is required if the validation is located on a form running on the server.
- **Use the NEW_FORM** instead of OPEN_FORM or CALL_FORM. Multiple forms open at the same time take more memory on the client. If another user has the same form open, it will share the memory on the server that is used by the program units. (No data portions or variable values of the form are shared.) However, if the user needs to return to an form that was previously open, you might consider using OPEN_FORM so that the reload does not require restarting the form, but just switching focus to it.

  One guideline is to use OPEN_FORM when memory is not a problem on the server and client and the network latency is high (i.e., the speed is slow). Use NEW_FORM when there is limited memory on the server and client and there is low network latency (a fast network).
- **Use subclasses** The "message diff-ing" feature of the Forms Services reduces the messages required to draw objects on the form by caching instructions for objects that share similar properties. For example, if you have an item that is 16 points high with an MS Sans Serif, 9 point font, the next item that the form needs to draw with those same characteristics will not require reloading the instructions from the server—they will be cached on the client because they were used once. Using default values for properties, subclassing (and SmartClasses), and visual attributes ensures that you will have standard settings that allow the message diff-ing feature to work.
- **Locate the Forms Services (server) physically close to the database server**. The reason is that the internal tuning that Oracle has performed on the Forms Services to client connection solves problems of network slowdowns. The database connection from Forms Services uses standard SQL*Net (Net8) mechanisms and this has no special tuning possibilities for Forms other than reducing the physical distance between machines.
- **Use PLL libraries** Although it may seem obvious to experienced Forms developers, it is worth pointing out that the use of PLL libraries enhances performance on the Web. Libraries are loaded into memory on the application server and that memory is shared by all instances of all forms that require the same code. Thus, even if you have one form that is used by many users at the same time, putting much of its code into a library will allow the concurrent users to share the memory space taken by the library. This means that you need to move the code out of the form into a library and then attach that library to the form. If you have many forms that are sharing the same library, the chance that more than one user will access that library increases along with the server memory savings.

Note

This author's paper "Forms Tuning Techniques—Users Will Sing Your Praise" presented at the IOUG-A Live 2001 conference (and available on the author's web site mentioned at the end of this paper) provides additional information on tuning your forms. Tuning forms for the Web must take into consideration all aspects of tuning, not just those aspects that are specific to web deployment.

5. Test on the Web

It almost goes without saying, but is important enough to state, that you should frequently test the application code you create on the Web (with the Run Form Web button at least) during the development process. It is a good idea to try a form on the Web before starting development, so you can be aware of the differences that will affect your design. Examine all types of items you will create. Test how complex you can make a form and have it operate efficiently. Check the action of GUI objects like buttons, check boxes, poplists, and radio groups. If you will be using complex objects such as the hierarchical tree control, be certain that the functions you require on the Web are supported by the object.

Note

Cursor icons may look different on the Web. Also, there are a number of extra icons available in web-deployed forms. Doc ID 68126.1 on Metalink (metalink.oracle.com) discusses the new cursor icons available (but not documented in the Forms help system).

6. Move Code to the Database

Placing as much code as possible on the database server is standard practice in most modern development efforts. It is important when you're deploying in the web environment for the same reason: any database-access code that is located on the client side (which is partially running on the application server) will create network traffic between the database server and that application server. The application server is already handling a number of functions and processes to support the Forms Server and Forms Runtime, and the fewer network transactions it has to handle, the faster it will work.

7. Set User Expectations

It is a good idea to get user buy-in to this technology early in the development cycle. This means that you have to have a web-deployed forms environment working that they can use to test the applications, but you will need that anyway for testing forms in development. If you create prototypes, be sure to show them using web-deployed forms. If you are not using the embedded MDI window feature of Developer R.6, stress to users that these applications run outside the browser in an external viewer. Early user buy-in will improve your chances for success.

8. Use the "Oracle Look and Feel"

If users are accustomed to client/server applications, they will find that forms running on the web look and act differently. Therefore, you may have to sell users on the slightly different look and feel of applications running on the Web. The problem is that the users may perceive the change as only a difference in the startup process and slight changes in the runtime look. However, users will experience other differences such as navigation timing and think that the application is not as fast as client/server applications. If you use the "Oracle Look and Feel" (OLAF) as shown in Figure 2, the application will look very different and this may signal to the users that a different look justifies a different navigational feel. Implementing this change only requires a command-line parameter (lookAndFeel=oracle). However, it drastically changes the form presentation layer and may be enough that users perceive this as a completely different (and better) operating environment.

This new look and feel uses rounded corners for window borders, scroll bars, buttons, poplists, LOV, and alert windows. There are also a number of color schemes that you designate with an HTML startup file parameter called colorScheme (with valid values of blue, khaki, olive, purple, red, teal, and titanium). The color schemes provide a certain set of coordinated colors as highlight and background colors. Figure 3 shows an OLAF alert. The alerts also have animated icons (the bell in this alert "rings" when the alert is shown).

9. USE A FORMS SHELL

Speed up your form testing by creating a form shell. This is a form with one text item and a button that executes a CALL_FORM to the form named in the text item. You keep this form running as you are developing and use it to test forms that you have compiled. This saves the time it takes to open the Forms Runtime executable each time you need to test a form. Instead of running the form from Form Builder, you press CTRL-C to compile the form, enter the name in the form shell, and click the button to run it. Although this technique is useful in client/server or in Web environments, it is particularly useful when you use the Web Previewer (Run Form Web button) or run the form in a browser as this runtime environment takes more time to start up. However, be aware that you can only run one Web Previewer at a time, and, if the Web Previewer is running, you cannot also run a form in client/server mode through the Form Builder. Figure 4 shows a sample forms shell running in the applet.

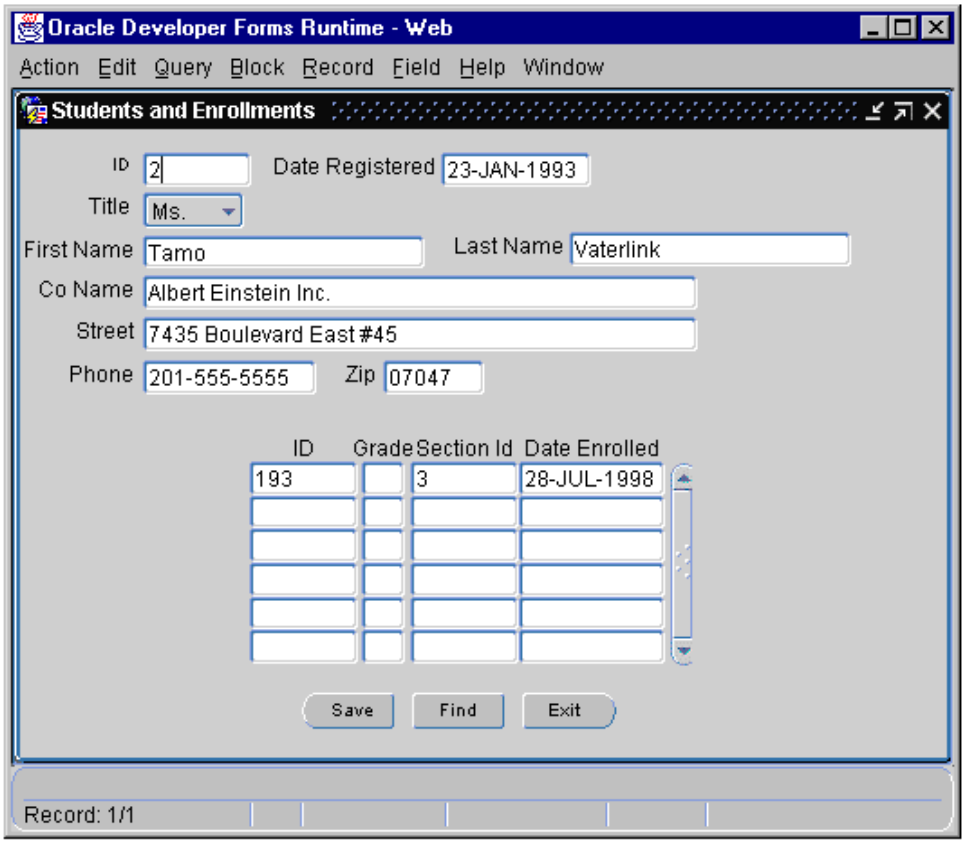

Figure 2. Form running with Oracle look and feel

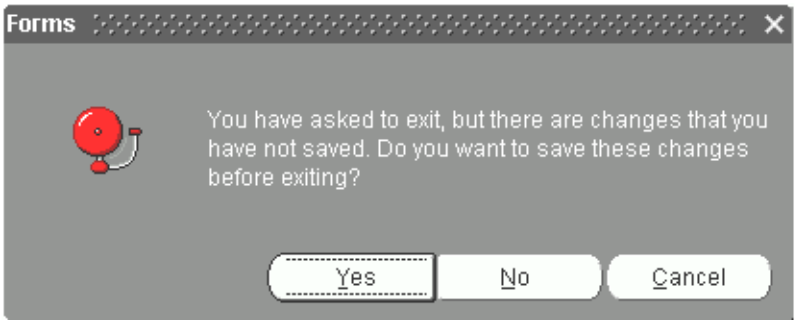

Figure 3. OLAF alert

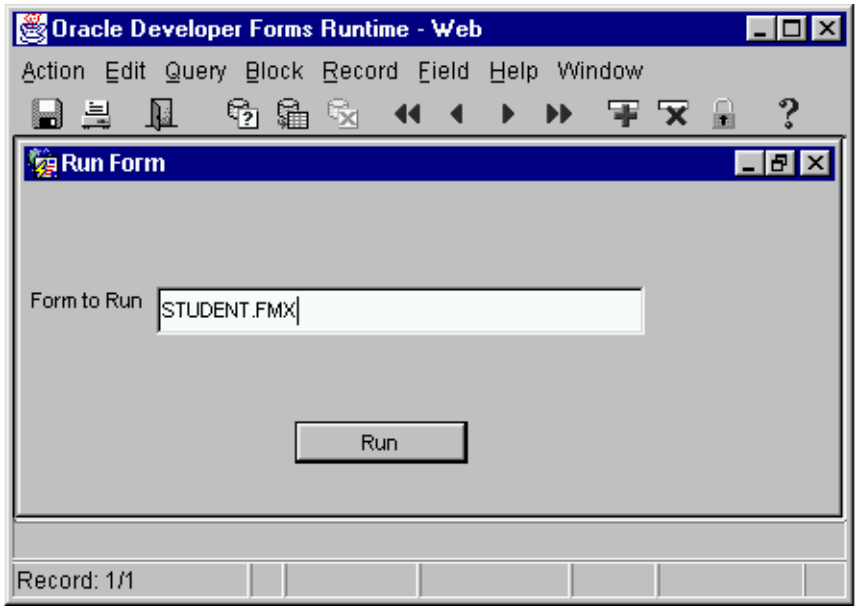

Figure 4. Forms shell on the Web

10. Understand Operating System Access

There are a number of considerations that arise because the environment is a Java Virtual Machine and the operating system for the Forms runtime resides on the application server middle tier. Since web-deployed forms run in a Java environment, the application server will process any calls to the operating system that your form makes. The client operating system is normally not available. This is a core difference from client/server that allows your form to access the files and operating system of the user's machine. The following Forms features or extensions are affected.

- **FFI** Foreign Function Interface (FFI) code extends the abilities of Forms by allowing you to call functions in external libraries (DLLs in Windows). Since "external" means the host operating system, FFI calls in web-deployed forms will be sent to the application server. If you are trying to call Windows DLL routines like calling WinHelp to show a help file, the call will be executed on the server, but the client will not see the results.

  This implies, too, that if you do use an FFI that does not require displaying something, the operating system might be different on client/server and web-deployed forms. For example, if you use an FFI call to a DLL that performs a complex calculation, you would create the DLL in Windows and register it to the FFI appropriately. However, if your deployment operating environment were a Solaris platform, the DLL would need to be recompiled under that operating system.

  Another example of a call to a DLL is the button hint feature introduced with Developer R.1. This relied on an FFI call to a DLL (through HINT.PLL) that would not work in web-deployed forms because it displayed on the client. Fortunately, Developer R.2 and R.6 use the *Tooltips* property on items and canvases to implement this feature, and these work in web-deployed forms.

- **OLE and DDE** Object Embedding and Linking (OLE) and Dynamic Data Exchange (DDE) have the same concerns as the FFI except that they are Windows specific. This means that even if you are using an OLE object that does not display, you cannot use it at all if your application server is running in a non-Windows operating system. For example, OLE is not available on Solaris, and you do not even have the option of recompiling for the new operating system.

- **OCX and VBX** OCXes (also called ActiveX) and VBXes (16-bit versions of OCXes) are Windows extensions written in a standard way and compiled into libraries. You can embed these in your form, capture events like button presses, and set properties programmatically. Since OCXes are usually display-oriented (such as the grid control that allows you to resize columns), they will not work in web-deployed forms—the form is running on the application server and cannot show the OCX in the client. You can use a JavaBean instead of an OCX if you can find or build one with the same functionality.

- **HOST** HOST calls in Forms create a separate session in the host operating system. In web-deployed forms, the HOST call will run the session on the application server. Therefore, if you want to perform some action on the client machine, you cannot do it with HOST. In addition, if you are using HOST in client/server to display something in the client browser, this will not work in web-deployed forms—the client browser cannot interact with the application server's environment.
- **User Exits** The user exit library extension must be compiled in the application server operating system and located in an accessible location on the application server. If the user exit has any display aspects, they will not appear in the client browser.
- **Windows Registry** The client Windows registry, like the system editor, is not available because Forms runs on the server. Changing or reading the registry using FFI calls to Windows DLLs will not work if the middle tier (application server) is running under a non-Windows operating system like Solaris.
- **Printing** from Forms is possible, but you have to realize that the printing will occur on the application server, not on the client machine. If the application server can access the client machine (through a network file system or other mechanism), you should be able to print from the application server to the client's printer. In addition, if you select **File→Print** in the applet window, you can print the screen on a client printer. Use ORARRP (available on MetaLink and OTN) for printing reports and forms locally.
- **TEXT_IO** This package provides functions that read and write files to the host's file system. As mentioned, in web-deployed forms, the host is the application server, so calls to TEXT_IO will access files on the application server. If the application server can see the client machine and has proper permissions, through a shared file system or other mechanism, it should be able to write files back to the client's file system. The ORARRP utility mentioned above allows TEXT_IO to work on the client machine. Remember that you can also use PJCs to access files as mentioned in category 4 above.
- **System Editor** Your forms can use a system editor, specified in the *Editor* property of an item, that normally calls an executable defined by the SYSTEM_EDITOR registry value. Since, in web-deployed forms, the form is running on the application server, the system editor is a program on the server. Calling this will, at best, open an editor on the server, not on the client. The workaround is to use a standard Forms editor. The client JVM supports this.
- **GET_APPLICATION_PROPERTY** If you have forms that will be deployed in different platforms (such as Web and client/server), you might have to write code that is different for each environment. You can check the runtime environment with the GET_APPLICATION_PROPERTY(USER_INTERFACE) call. This will return a value of "WEB" if the form is running under web-deployed forms. If you are running in MS Windows, the function will return "MSWINDOWS32." You can test for the operating system and conditionally execute procedures that the particular operating system requires.
- **WEB.SHOW_DOCUMENT** This Forms built-in can connect to another URL and run another web application, or just show a web page in the browser.

> **Note**
>
> As mentioned in category 4, PJCs can provide much of the functionality of the Windows-specific strategies such as FFI and DDE. The drawback is that you need to find (or write) a replacement for the features.

11. Know the Limitations and Workarounds

There are some known feature restrictions with web-deployed forms due to the current state of the Java language. There are also some features that work differently or not at all; some of these are documented and some are not. If you are moving existing forms to the Web or if you are designing new forms, it is good to keep the limitations in mind. There are two things to remember when designing Forms for Intranet deployment in general:

- **Web-deployed forms run on the application server** and present only the user interface to the client.

- **This is a Java environment,** not a Windows environment. The look and feel is different between these two environments.

TRIGGERS THAT DON'T FIRE

There are a few triggers that do not fire in web-deployed forms. Avoid using these even though they work in client/server applications. These triggers are WHEN-MOUSE-LEAVE, WHEN-MOUSE-MOVE, and WHEN-MOUSE-ENTER. These don't fire because of a limitation in Java.

MAGIC MENU ITEMS

Avoid using magic menu items such as Cut, Copy, or Paste, because they do not work in web-deployed forms. Use CUT_REGION, COPY_REGION, and PASTE_REGION instead. The CTRL- key combinations (CTRL-X, CTRL-C, and CTRL-V) do work in web-deployed forms, however.

MAXIMIZING THE WINDOW

The call to SET_WINDOW_PROPERTY for maximizing or minimizing the Forms MDI window does not work in web-deployed forms. Therefore, you have to resize the window by setting the X and Y sizes and locating the form at position 0, 0 by using SET_WINDOW_PROPERTY. Minimizing the MDI window also does not work in web-deployed forms.

Maximizing an internal window (non-MDI window) in web-deployed forms maximizes the window. However, the window title is not absorbed into the menu bar as it is in client/server. This means that if you use the extra canvas area that results from the window title disappearing in client/server, your layout may need to be adjusted when you move the form to the Web because that extra canvas area does not appear when you maximize the inner window.

CANVASES

In releases before R.6, stacked canvases sometimes required explicit SHOW_VIEW and GO_ITEM calls to display, even though they might not require this in client/server. There may be other strange behaviors with stacked canvases that can be fixed if you use SHOW_VIEW to explicitly display the canvas. Another thing to try is to increase the space between stacked canvases if one is being automatically hidden. Be sure to check the forms on the Web, because you may not see these as issues in a client/server environment.

SYSTEM MOUSE VARIABLES

You can use the system variable :SYSTEM.MOUSE_BUTTON_SHIFT_STATE to return a character string in a mouse trigger indicating which shift key was clicked (ALT, CTRL, or SHIFT). The help system states that the values returned in these variables are operating-system specific. Since web-deployed forms run in a Java environment, you might expect the values to be different and they are. In the client/server Windows environment, the strings returned by this variable are "Ctrl+", "Alt+", and "Shift+", indicating that the corresponding button was pressed when the mouse button was clicked. On the Web, the values are "Control+", "Alt+", and "Shift+".

Another system variable, :SYSTEM.MOUSE_BUTTON_PRESSED, returns a character string indicating which mouse button was pressed. The normal values (in client/server) are "1", "2", or "3" for the left, middle, and right mouse buttons, respectively. On the Web, only the left button returns a value (at least in version 6.0).

Another variable, :SYSTEM.MOUSE_BUTTON_MODIFIERS, provides an operating-system-independent value. You do not need to test the GET_APPLICATION_PROPERTY when checking the mouse button modifier keys. The values returned by this variable are "Shift+", "Caps Lock+", "Control+", "Alt+", "Command+", "Super+", and "Hyper+". Look up MOUSE_BUTTON_MODIFIERS in the help system for more information. If you are creating new forms (not just converting existing client/server forms to the Web), you can use this variable.

12. TROUBLESHOOT PROBLEMS EFFICIENTLY

When you are troubleshooting Forms on the Web, there may be some frustration because an operation may work fine in client/server but not on the Web. One of the main differences with web-deployment is that the application server is involved in the process of starting and running the form. Therefore, if you have determined that a problem you are troubleshooting is not caused by the form, or the database, you may need to troubleshoot the application server. The following techniques apply to troubleshooting a Forms Server running on Oracle Application Server. If you are using Oracle9i application server, the considerations are similar, but the terms and techniques may vary slightly.

> **Note**:
>
> Forms 9i promises the feature of remote debugging. For more information, refer to the white paper “Oracle 9iAS Forms Services - Best Practices for Application Development” available in the Forms Developer section of the OTN web site (otn.oracle.com).

JAVA CONSOLE

You can turn on a message box that displays the output of various steps the Forms Server and Java applet go through when loading a form. When the Java Console is turned on, you will see a message window as in Figure 5.

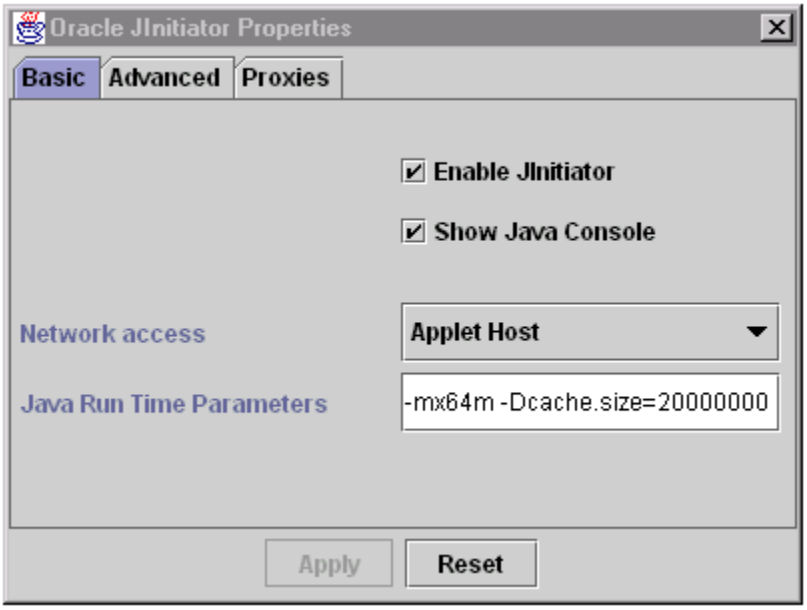

Figure 5. The Java Console

You can turn this console on using the "Oracle JInitiator Control Panel" application that is installed into your Windows start menu when you install JInitiator. Running this application displays the dialog as Figure 6 shows.

Checking the checkbox for "Show Java Console" will display the console when you start the next form from the browser. While you will still have to do some digging to determine the cause of the problem, this message window can supply error message numbers to check and a sense of what is working and what is failing.

TIPS FOR NETWORK TESTING

- **Use the TCP/IP ping utility** to check the health of the network, using the following command:
  `ping -s myhost.corp.com 1472 -5`
  The -s specifies that a large packet size will be tested; 5 indicates that the test will be made five times; 1472 indicates the size of the tested data transmission. The output from this command will indicate how well the network is transmitting data.

- **Use the utility netstat** to determine how the network is set up. Issuing a command line command for netstat -I, you will see a list of various measurements. If the collisions (shown in the Collis column) are high (greater than 1 to 2% of the total packets reported in the Opkts column), the network is busy. If the packets queued (in the Queue column) are not zero, you have data that cannot be transmitted. Both conditions may require extra work and help from the network administrator to resolve.

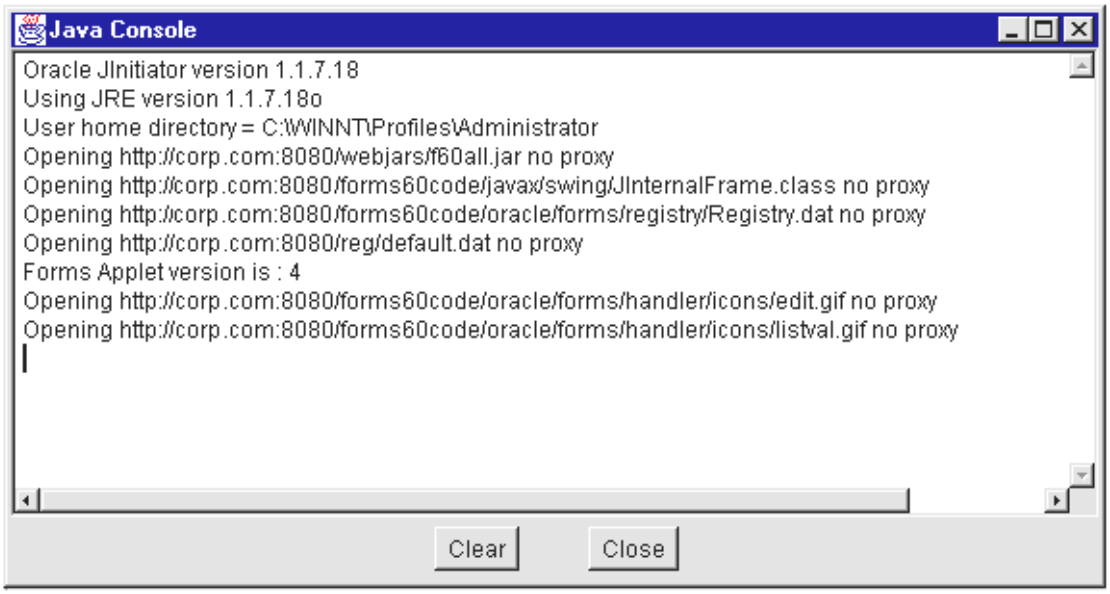

Figure 6. The JInitiator Control Panel

TIPS FOR THE FORMS SERVER

The following tips apply to the Forms Server process. Some also apply to the Forms Servlet strategy introduced with Forms 6i.

- **Determine the state of the Forms listener** by accessing the appropriate UNIX machine. Do this by issuing "ps –ef | grep f60" from the UNIX command line. If the listener port is not running then start it using the procedures outlined above. Otherwise, stop and start the listener and Forms Server if the process seems to be hung.
- **Review the details of FRM-99999** With Forms 6i, the FRM-99999 error now contains more details. For example, if the error is FRM-99999, Error 1412, you have programmatically tried to set the position of the scrollbar on a block that has no scrollbar (using the SET_BLOCK_PROPERTY built-in). If you tried to get the scrollbar position information from a block that had no scrollbar, you would receive an error 1413 with the 99999. The release notes (relnotef.pdf) contains a more extensive list of error messages that are associated with FRM-99999.
- **Check the directory listing for webicons** If the Forms Server has problems finding the toolbar icons, check the directory listing for webicons. Also, check the icons directory listed in the registry.dat (in the FORMS60 directory on the server) or default.dat (also located on the application server). There should be an entry for "default.icons.iconpath= /webhtml/icons/" or something similar to that.
- **Test whether the JAR files are being cached** Enter the following parameter in the Java Console runtime parameters field:

  ```
  -Dcache.verbose
  ```

  This will add messages that are specific to the Forms Server to the Java Console window (described above). If you want to capture the output of this process to a file, use the following parameter in the Java Console:

  ```
  -Dcache.logfile mylog.txt
  ```

  This will output the results of the Java Console into mylog.txt. In the Advanced tab of the control panel, you can check the checkbox for "Enable Debug" to get additional messages.

- **Use other logging facilities** Activate these by providing a parameter to the Forms Server processes when you start them on the command line. The parameter is "log=*logfile*" where *logfile* is the name of the file that you want to have messages stored in. This technique can be used for the server (f60svrm or ifsrv60) and the control program (f60ctl).

- **Look for stack trace files** These files are generated on the client machine when an error occurs. Look for files with an .RPT extension. The names of those files will indicate which program caused the error (for example netscape.rpt means that the Netscape browser had an error).
- **Fix broken images** Images not rendering properly may be due to the FORMS60_MAPPING environment variable not being set properly. This is mentioned in the Environment Variables section above, but it is important that you use the proper slashes ("/" for a virtual directory) and place them on both sides of the virtual directories.

13. DON'T GO IT ALONE

If you have been working primarily in the client/server Forms world, you will find the web-deployed Forms world slightly different. As this paper has described, there are new concerns and issues that you will have to address. Some of the workarounds and solutions that you have applied to the client/server Forms world do not apply and you will have to develop new workarounds and solutions. A strong recommendation is to take advantage of the collective knowledge base of Forms developers. Others who are working in the same environment may have already answered problems and questions that you might have.

The following is a list of potential resources for networking with fellow Forms-developers. You will find them an invaluable resource. Also, they will give you an opportunity to add to this knowledge base by sharing your solutions and advice with other users.

- **IOUG discussion forums** www.ioug.org These online discussion areas are free to members and non-members alike. There is a special discussion forum that addresses development tools and you will find users who can assist in answering questions about web deployment.
- **ODTUG list serve** www.odtug.com (look for Electronic Lists and sign up there.) The Oracle Development Tools User Group focuses on development tools such as Forms. When you sign up for the Developer list, you will get email from all who contribute to the list and will be able to ask questions that show up in the list members' email. The lists are open to non-members.
- **Oracle Technology Network discussion forums** otn.oracle.com Oracle Technology Network (mentioned in category 2 above) is a key developer and DBA resource. It also offers active discussions from users and feedback and contributions from Oracle's product managers.
- **Your local Oracle users group** Be sure to take advantage of the networking possibilities of your local Oracle users group meetings and conferences. Contact the IOUG office (ioug@ioug.org) if you need help finding the group nearest to you. Meetings are great places to contact and exchange ideas with users who are working in the same environment.
- **Training** Oracle and third-party vendors offer courses in Forms basics and advanced techniques for best practices that can help you gain knowledge about web deployment, the Forms Servers, and Form Builder. These courses give you a head start when venturing into a new environment such as the Web.

CONCLUSION

When you work with web-deployed forms, there are a number of things to think about that you do not have to think about in client/server forms. This paper has supplied tips for these things as well as for some details on how you can best design, develop, and deploy applications in this environment. With this kind of knowledge, you will be able to successfully implement Forms applications that are run using web technology. The best final advice is to network with other user group members to compare proven techniques and solutions to problems.

About the Author

Peter Koletzke is a Technical Director and Principal Instructor for the Enterprise e.Commerce practice at Quovera (formerly Millennia Vision), in Redwood Shores, CA. Peter is Executive Vice President and Director of Web Content for the IOUG and columnist for the ODTUG Technical Journal. He is a frequent speaker at various Oracle users group conferences where he has won awards such as Pinnacle Publishing's Technical Achievement, ODTUG Editor's Choice, and the ECO/SEOUC Oracle Designer Award. At Oracle OpenWorld in December 2001, Oracle University presented him with one of the first six Oracle Certified Master's awards. He is coauthor, with Dr. Paul Dorsey of the Oracle Press (Osborne McGraw-Hill) books: *Oracle JDeveloper 3 Handbook*, *Oracle Developer Advanced Forms and Reports*, *Oracle Designer Handbook*, *2nd Edition*, and *Oracle Designer/2000 Handbook*. http://ourworld.compuserve.com/homepages/Peter_Koletzke and

http://www.quovera.com

Quovera provides strategy, systems integration, and outsourced application management to Fortune 500, high-growth middle market and emerging market companies. The firm specializes in delivering intelligent solutions for complex enterprises, which improve productivity within the customer's business and optimize the customer's value chain, through integration of its customers and suppliers. The company also outsources the management of "best of breed" business applications on a recurring revenue basis. Quovera refers to its business model as "Intelligent – Application Integration and Management. http://www.quovera.com

Appendix A – The Architecture

Web-deploying your Forms applications uses a mix of Web and client/server. In essence, it is a three-tier environment that is really just an evolution of an old environment—that of two-tier or client/server. Client/server architecture consists of two logical machines: a client and a server (although you can run both client and server processes on the same machine). The client runs the runtime software and accesses the Form Builder files on the local machine or on a file server. The client is "thick," that is, it must have enough hard disk space and RAM to store and run the Forms runtime files and the forms. The server runs the database management system, which also accesses the data files. Figure A shows the components of a simple Oracle Forms Developer client/server runtime environment. In this model, the client machine accesses application runtime files on its disk system (or on a file server disk system). The runtime program operates on the client machine. This communicates with the database server for its data needs.

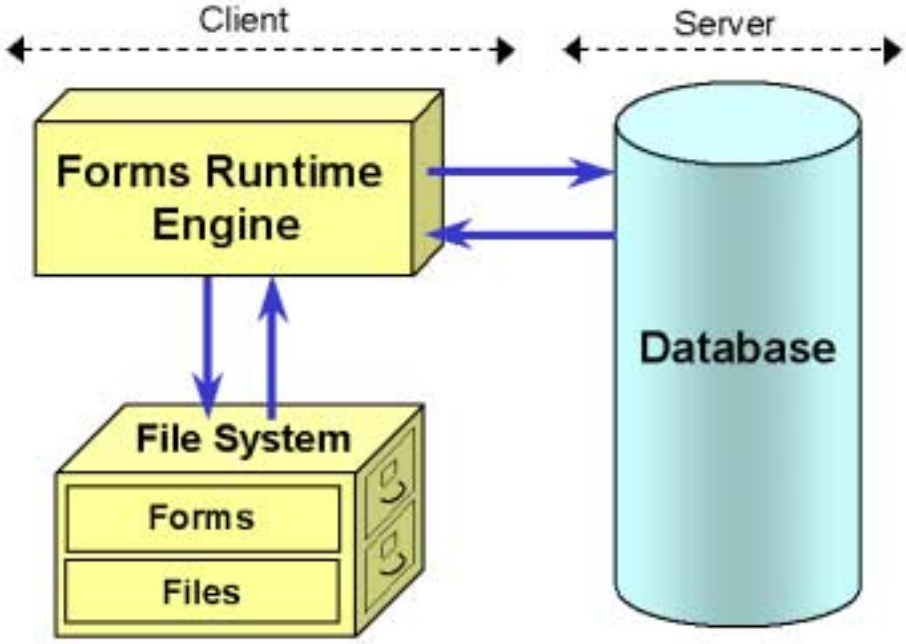

Figure A: Client/server Forms runtime environment

The web-deployed forms multi-tier (or web-enabled client) architecture consists of three logical machines: a client, an application server, and a database server. Normally, each tier is located on a separate machine. It is possible to run the tiers on one or two machines, but maintenance and performance optimization are easier if the tiers are on separate machines. Figure B shows the components of the web-deployed three-tier architecture.

THE JAVA VIRTUAL MACHINE

The client runs a browser and, in the case of web-deployed forms, the browser runs a Java Virtual Machine (JVM) supplied by JInitiator. *JInitiator* is the Oracle version of a Sun Microsystems browser plug-in. The plug-in extends the capabilities of the browser so it can present the interface to a form running on an application server. JInitiator takes input from the client as well, but most of the processing, such as validation and trigger code, executes on the application server or database server. It contains the results of network optimizations that Oracle has built into the code.

JInitiator contains the same functionality as AppletViewer (an earlier version of the Oracle JVM). Both JInitiator and AppletViewer are pure Java solutions and rely heavily on Sun's Java implementations. Oracle has certified Microsoft Internet Explorer (IE) Release 5.0 as a stand-alone environment for web-deployed forms with no plug-in (JInitiator) requirement.

JInitiator offers some benefits over the native (IE 5.0) browser support, such as *on-demand JAR files*, which load Java class archive files when required instead of at startup, and *persistent JAR file caching*, which saves JAR files onto the disk so they can be used from a local installation instead of from a network installation. This saves download time. JInitiator offers the level of support required for running Oracle Applications on the Web.

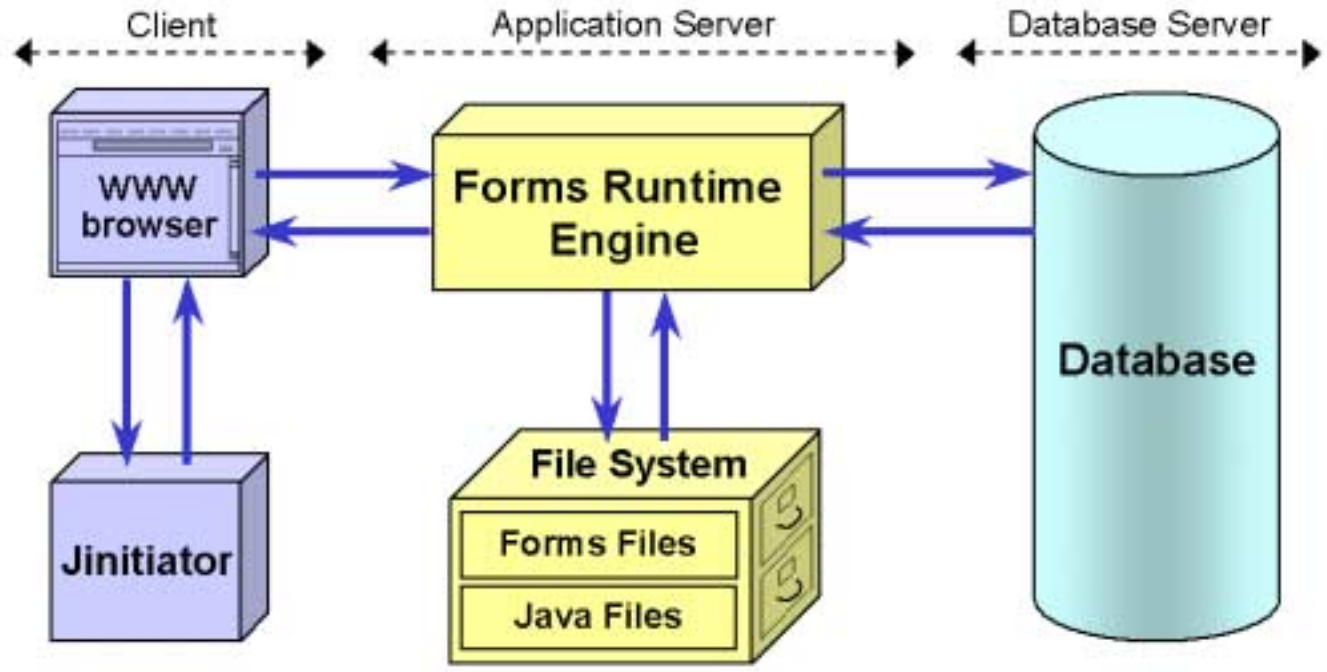

Figure B: Multi-tier web-deployed Forms architecture

THE MIDDLE TIER

The Forms runtime engine running on the middle tier (application server) accesses the file system on the server (or another networked location) and presents the user interface layer in a Java applet on the client tier. The application server in this environment is essentially the same as the client tier of the client/server environment. It runs the runtime engine (called *Forms Services*) that is a separate process or Java servlet and accesses the file system for Forms files and the database for data. It is critical to remember that the form presented in the client's browser is actually running on the middle tier with only the presentation layer revealed on the client's side.

The client is considered "thin" from a software standpoint because it does not need to store the Forms runtime programs or the Forms application files. The application server stores the runtime files and a Forms runtime process or servlet process (shown in Figure B as the runtime engine). There are other web components that also run on the middle tier to start a runtime

session: the Forms Server and the HTTP listener (Oracle9*i* application server or any non-Oracle web server such as Netscape Enterprise Server, Microsoft Internet Information Server, or Apache). The Forms Server and the HTTP listener are not used while a form is running, although the Forms Server is used to start the form. The database server runs the database software to manage the database files the same way it does in the client/server model.

The only real difference between the client/server and a multi-tier model is that the multi-tier client is split into two machines: the application server tier and the client. The database server is the same in client/server and multi-tier models.

THE RUNTIME MODEL

There are a number of components to the web-deployed forms runtime environment. The best way to understand what the components are responsible for and how they work together is to examine the lines of communication during the two main events: startup and runtime.

STARTUP STEPS

The first step in starting a form is the client request, via a URL (uniform resource locator), to the application server for a startup HTML file. The *startup HTML file* contains special HTML tags that start the JVM and ensure that the proper Java class files are available for the session. There are examples of the HTML startup file in the online documentation as well as in the Forms directory. The file uses different tags for Internet Explorer and for Netscape but you can embed both commands in a single file. The web listener on the application server determines that the URL indicates a standard static HTML file, retrieves the HTML from the file system, and sends it to the client browser.

The Forms 6i install loads two types of files that are used when starting up the first form. These are loaded into the forms60/admin/server directory. The first type of file is the startup file, which has two manifestations. BASE.HTM (used for appletviewer or native browser applets) and BASEJINI.HTM (used when JInitiator is used to display the form) contain values that affect the startup, some of which are specified as replaceable parameters. These files work the same way as the STARTUP.HTML file described above, but they work with the second type of file, the configuration file, to fill in the values of the various startup parameters that can be used in this environment. The configuration file, called FORMSWEB.CFG, contains the same type of parameters. Values that are assigned to the parameters in this file are used for the replaceable parameters in the startup files.

The sample HTM files that are used should be complete enough that you would not need to change them. Oracle recommends that you make changes to the parameter values in the configuration file. This will allow you to keep different configuration files for different applications and customize the parameters for each application.

There is more information on the contents and workings of these files in the OIN document *Deploying Forms Applications to the Web with Forms Server* document (**Help→Manuals** in the Form Builder).

The next set of steps handles the startup of the Forms runtime and the form itself. The tags in the HTML file indicate that JInitiator will be used to display the form through the standard plug-in extension mechanism provided by the browser. They also indicate the Java class files that are required by the Java applet and that the applet start a connection with the Developer Forms Server listener (through the serverArgs and serverPort parameters). This information in the HTML file is sent to the web listener, which passes the request for a connection to the Forms Server Listener so that the Forms Server Listener can establish the connection with the client.

The Java class files are retrieved from the application server (if they have not been cached on the client machine) and sent to the client browser. The browser starts the Forms client applet session as a separate window or as a window embedded in the browser. The Forms client contacts the server to request a runtime session. The Forms Server Listener then starts up a Forms Runtime Engine or Forms servlet session.

RUNTIME STEPS

The Forms Server Listener breaks the connection between the web listener and browser and sets up a TCP/IP socket connection directly from the Forms Runtime Engine to the browser. The form running in that session accesses forms and other files in the file system and communicates with the database through a standard SQL*Net or Net8 connection. The Forms client session is a child process of the browser session.

THE DEVELOPMENT ENVIRONMENT

Generally, the development environment for web-deployed forms is a client/server environment. Forms Builder 6i offers a Run Form Web button that you can use to run the file in the Forms applet client. This client is a JVM that runs in a client/server environment and uses the same Java classes as the web environment JVM uses. This allows you to check the look and feel of the form as it would appear on the Web. In addition to this test, you should periodically move the form to the application server, recompile it (if it is a different operating system), and run it from your browser. This process will ensure that the application server components handle the form in the same way that the client JVM does and that all objects will be handled in the same way.

Another potential development technique is to save and run the form on the application server. If you are running Windows NT, the .FMB and .FMX files you are working with can reside on the server, and you can access them through the file server network. If you are running an X-terminal or X-terminal emulator for a Solaris server, Form Builder will reside on the file server, and you will run a terminal session to access it. The forms files themselves can then be saved to the (development) application server so you can run the file immediately after compiling it in the X-terminal Form Builder.

This is really the same strategy as the client/server method, except that you are not developing on the client and do not need to have the Form Builder installed locally. The terminal strategy eliminates the need to copy the file to the application server for occasional testing.